Computing and Informatics, Vol. 23, 2004, 239–254

DRAWING FREE TREES INSIDE SIMPLE POLYGONS USING POLYGON SKELETON*

Alireza Bagheri, Mohammadreza Razzazi

Software Systems R & D Laboratory
Department of Computer Engineering & IT
Amirkabir University of Technology (AUT), Tehran, Iran
Institute for Studies in Theoretical Physics and Mathematics (I. P. M.)
e-mail: bagheri@ce.aut.ac.ir, razzazi@ce.aut.ac.ir

Manuscript received 26 May 2003
Communicated by Branislav Rovan

Abstract. Most of graph drawing algorithms draw graphs on unbounded planes. In this paper we introduce a new polyline grid drawing algorithm for drawing free trees on plane regions which are bounded by simple polygons. Our algorithm uses the simulated annealing (SA) method, and by means of the straight skeletons of the bounding polygons guides the SA method to uniformly distribute the vertices of the given trees over the given regions. Our results show improvements to the previous algorithms that use the SA method to draw graphs inside rectangles. To our knowledge, this paper is the first attempt for developing algorithms that draw graphs on regions which are bounded by simple polygons.

Keywords: Graph drawing, simulated annealing, straight skeleton

1 INTRODUCTION

Graphs are well-known structures that have many applications. Hence drawing graphs "nicely" has been investigated by many researchers. There are some aesthetics for nice drawing of graphs that are mentioned in the literature. Some of the most important aesthetics are: *minimizing the number of edge crossings*, *minimizing*

* This research was in part supported by a grant from I. P. M. (No. CS1383-4-1)

the number of bends per edge, *increasing symmetry of drawing*, *maximizing angular resolution*, and *uniform distribution of vertices* over drawing regions [7, 16].

Most of graph drawing algorithms draw graphs on unbounded planes and few of them draw graphs on regions which are bounded by rectangles. However, there are some applications in which it is desirable to draw graphs on regions which are bounded by general polygons. For example, consider a graphical user interface in which we would like to draw a graph inside a star-shaped polygon to make it more beautiful. In this paper we consider polyline grid drawing of free trees inside simple polygons by means of the straight skeletons of polygons [1, 2, 3, 14, 15] and the simulated annealing (SA) method. We assume that the reader is familiar with the SA method.

The edges of the given tree may have some bends in our drawings if the given region is bounded by a non-convex polygon. When drawing large graphs, drawing results of algorithms that use clustering [10, 11, 12] usually have much fewer edge crossings than drawing results of those that do not use it [4, 6, 13, 17, 18, 19]. Our algorithm uses a special form of clustering and uniformly distributes the vertices of the given tree over the given region.

We compare our drawing results to those of the algorithm introduced in [9], which we call the SA algorithm. The SA algorithm draws graphs inside a given rectangle and similar to ours uses the SA method. Because of employing geometrical properties of drawing regions by our algorithm, our drawing results show more symmetry than the results of the SA algorithm. The average number of edge crossings in drawings of some complete binary trees inside a square, both by our algorithm and the SA algorithm, is given in Table 1. This result has been obtained from running our algorithm and the SA algorithm 100 times for each tree on a 640×480-grid. As can be seen, our algorithm also outperforms the SA algorithm in the number of edge crossings. In Section 2, the straight skeleton is briefly described. In Section 3, our algorithm is introduced. In Section 4, some drawing results of our algorithm are illustrated and compared to the drawing results of the SA algorithm. In Section 5, the conclusion is stated.

Trees	7-node	15-node	31-node	63-node	127-node
The SA alg.	0	0	0	1.88	34.1
Our alg.	0	0	0	0	1.85

Table 1. The average number of edge crossings in drawings of some complete binary trees inside a rectangle both by our algorithm and the SA algorithm

2 THE STRAIGHT SKELETON

There are two types of skeletons for simple polygons—*the medial axis* and *the straight skeleton*. The medial axis of a given simple polygon P consists of all the interior points whose closest point on the boundary of P is not unique [8]. While the medial

axis is a Voronoi-diagram-like concept, the straight skeleton is not defined using a distance function but rather by an appropriate shrinking process.

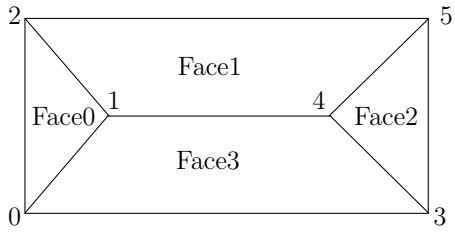

Fig. 1. The straight skeleton of a rectangle

The straight skeleton is defined as the union of the pieces of the angular bisectors traced out by the polygon vertices during the shrinking process [1, 2, 3, 14, 15]. The straight skeleton, in general, differs from the medial axis. If P is convex then both structures are identical; otherwise, the medial axis contains parabolically curved segments around the reflex vertices of P, which are avoided by the straight skeleton. In this paper, we consider drawing of free trees on plane regions which are bounded by general simple polygons, and to avoid parabolically curved segments we use the straight skeletons as the skeletons of polygons. In the following, by the skeleton we mean the straight skeleton. The skeleton of a given n-gon P partitions the interior of P into n connected regions which are called *faces*. Each face is related to just one edge of P. The bisector pieces are called *arcs* and their endpoints are called *nodes*. When P is simple, the structure is tree. Figure 1 shows the straight skeleton of a rectangle.

3 OUR DRAWING ALGORITHM

In this section, after introducing some new definitions, we explain our drawing algorithm (from now on, for simplicity, we use trees for free trees and polygons for simple polygons). Our algorithm produces a polyline grid drawing of the given free tree which is bounded by the given polygon. All the vertices and the bends are put on the grid points, but the edge crossings are not restricted to be on the grid points.

Definition 1. For a skeleton S and its node i, let $FaceSet(i; S)$ be the set of all the faces which have node i as a vertex on their boundary.

Example 1. Consider skeleton S of Figure 1, we have $FaceSet(0; S) = \{\text{Face0}, \text{Face3}\}$ and $FaceSet(1; S) = \{\text{Face0}, \text{Face1}, \text{Face3}\}$.

Definition 2. For a tree T and its edge (i, j), let $CloserSet(i|j; T)$ be the set of all the nodes of the tree whose graph-theoretic distance from node i is shorter than from node j.

Example 2. Consider skeleton S of Figure 1, $CloserSet(1|4;S) = \{0,1,2\}$ and $CloserSet(4|1;S) = \{3,4,5\}$.

By considering the tree structure of the skeletons, this definition is also applicable to the skeletons. In the following we describe each step of the algorithm in details. The pseudo-code of our algorithm is as follows.

Free Trees Drawing Algorithm

input: a free tree T, a simple polygon P, and a 2D grid G

output: a polyline grid drawing of T on G which is bounded by P

Step A. Computing the polygon skeleton and the area of the faces

Step B. Computing the weights of the nodes of the skeleton

Step C. Computing the weights of the edges of the skeleton

Step D. Computing the weights of the edges of the tree

Step E. Mapping the tree onto the skeleton

Step F. Removing possible crossings between the edges of the tree and the polygon

Step G. Drawing the tree using the SA method

Step A. Computing the polygon skeleton and the area of the faces

Let us call the boundary of a face Pface. We obtain straight skeleton S of given polygon P using [8]. Since all the Pfaces are simple polygons we can use the following formula to compute the area of face f [5].

$$Area(f) = \frac{1}{2}\sum_{i=0}^{n-1}(X_iY_{i+1} - X_{i+1}Y_i)$$

Here (X_i, Y_i) are the coordinates of vertex i $(i = 0, \ldots, N-1)$ of the Pface of face f; $X_n = X_0$ and $Y_n = Y_0$. To use the above formula, the vertices of the Pfaces should be ordered clockwise or counter clockwise.

Step B. Computing the weights of the nodes of the skeleton

For each node i of skeleton S we compute the following sum as the weight of node i.

$$W_{IA}(i;S) = \sum_{f \epsilon FaceSet(i;S)} Area(f)$$

$W_{IA}(i;S)$ denotes the weight of the incident area of node i of skeleton S. In fact, this weight represents the total amount of the area of the faces which are incident to vertex i of skeleton S.

Step C. Computing the weights of the edges of the skeleton

For each endpoint of every edge (i, j) of skeleton S, we consider a weight. The difference of these two weights is considered to be the weight of edge (i, j) and is denoted by $W_E((i, j); S)$. $W_{CS}(i|j; S)$ is defined as sum of the weights of the nodes belonging to $CloserSet(i|j; S)$.

$$W_{CS}(i|j; S) = \sum_{m \epsilon CloserSet(i|j;S)} W_{IA}(m; S)$$

$$W_{CS}(j|i; S) = \sum_{m \epsilon CloserSet(j|i;S)} W_{IA}(m; S)$$

$$W_E((i, j); S) = |W_{CS}(i|j; S) - W_{CS}(j|i; S)|$$

Step D. Computing the weights of the edges of the tree

For each endpoint of every edge (i, j) of tree T, we consider a weight. The difference of these two weights is considered to be the weight of edge (i, j). By $|S|$ we mean the cardinality of set S. If for each node i of tree T we define $W_{IA}(i; T) = 1$, we have:

$$W_{CS}(i|j; T) = \sum_{m \epsilon CloserSet(i|j;T)} W_{IA}(m; T) = |CloserSet(i|j; T)|$$

$$W_{CS}(j|i; T) = \sum_{m \epsilon CloserSet(j|i;T)} W_{IA}(m; T) = |CloserSet(j|i; T)|$$

$$W_E((i, j); T) = |W_{CS}(i|j; T) - W_{CS}(j|i; T)|$$

Definition 3. By *the middle edge* of a skeleton S (tree T) we mean an edge of S (T) that has the minimum weight among all the other edges of S (T).

Definition 4. A skeleton (tree) may have more than one middle edge. It is guaranteed by Lemma 1 that in this case these edges share an endpoint which is called *the middle node*.

Definition 5. By *the middle-connected node* we mean a node that is connected to the middle node by an edge.

Step E. Mapping the tree onto the skeleton

In this step, we are going to uniformly distribute the vertices of the tree over the given region. We do this by means of a recursive mapping procedure which recursively maps the middle edge or the middle node of the tree onto a proper

point of the skeleton. This mapping procedure provides a mapping list of the nodes of the tree which are mapped onto the corresponding points of the skeleton. This mapping list is used by the SA method to distribute the vertices of the tree over the given region. The termination condition of the procedure is satisfied when the number of the nodes of the given tree or the number of the edges of the skeleton becomes less than one. Considering the weighted skeleton and the weighted tree, four cases are possible in each call of the mapping procedure:

Case 1. Neither the tree nor the skeleton has a middle node.

Case 2. The tree and the skeleton both have a middle node.

Case 3. The tree has a middle node, but the skeleton does not have a middle node.

Case 4. The tree does not have a middle node, but the skeleton has a middle node.

Case 1. In this case the tree and the skeleton both have a middle edge but not a middle node. We map the middle edge of the skeleton onto the middle edge of the tree and omit these two middle edges from the skeleton and the tree. This divides the tree and the skeleton into two sub-trees and two sub-skeletons, respectively.

We should add a record to the mapping list, to show that the middle edge of the tree is mapped onto the middle edge of the skeleton. We substitute the middle edge of the tree with a path of length two whose extreme vertices are the endpoints of the middle edge and its internal vertex is a dummy vertex. Then we record in the mapping list that this newly added dummy vertex of the tree is mapped onto the middle point of the related middle edge of the skeleton. After termination of the algorithm, these dummy vertices may appear as some bends in the edges of the tree.

Then we update the weights of the edges of the two sub-skeletons and the two sub-trees. To do this, consider each sub-skeleton (sub-tree) as a directed tree whose root is the endpoint of the omitted middle edge that is attached to this sub-skeleton (sub-tree), and the edges are directed from the root toward the leaves. Suppose edge (v, u) is the omitted middle edge of skeleton S (tree T). The following pseudo-code shows how the weights of the edges of each sub-skeleton of skeleton S are updated. We can use this pseudo-code to update the weights of the edges of each sub-tree of tree T by replacing S with T. Then we apply the mapping procedure recursively for each sub-skeleton and its related sub-tree.

for each directed edge (i, j) *of each sub-skeleton rooted at* u *do*
{
$W_{CS}(i|j; S) = W_{CS}(i|j; S) - W_{CS}(v|u; S)$
$W_E((i, j); S) = |W_{CS}(i|j; S) - W_{CS}(j|i; S)|$
}

for each directed edge (i, j) *of each sub-skeleton rooted at* v *do*
{
$W_{CS}(i|j; S) = W_{CS}(i|j; S) - W_{CS}(u|v; S)$
$W_E((i, j); S) = |W_{CS}(i|j; S) - W_{CS}(j|i; S)|$
}

Case 2. In this case the tree and the skeleton both have a middle node. We omit the middle node and its incident edges from the tree. For the skeleton we just disconnect edges that are connected at the middle node. So the skeleton and the tree are divided into two or more sub-skeletons and sub-trees, respectively. We record in the mapping list that the middle node of the tree is mapped onto the related middle node of the skeleton.

Then we update the weights of the edges of the sub-skeletons and the sub-trees. To do this, consider each sub-skeleton as a directed tree whose root is the middle node and each sub-tree as a directed tree whose root is a middle-connected node, and the edges are directed from the root to the leaves. Suppose node u is the middle node of skeleton S (tree T). The following pseudo-code shows how the weights of the edges of the sub-trees and the sub-skeletons are updated.

$SUM_T = \sum_{(u,i)\epsilon T} W_{CS}(i|u; T)$

for each directed edge (i, j) *of each sub-tree rooted at middle-connected node* v *do*
{
$W_{CS}(i|j; T) = W_{CS}(i|j; T) - SUM_T + W_{CS}(v|u; T) - 1$
$W_E((i, j); T) = |W_{CS}(i|j; T) - W_{CS}(j|i; T)|$
}

$SUM_S = \sum_{(u,i)\epsilon S} W_{CS}(i|u; S)$

for each directed edge (i, j) *of each sub-skeleton which is rooted at middle node* u *and includes middle-connected node* v *do*
{
$W_{CS}(i|j; S) = W_{CS}(i|j; S) - SUM_S + W_{CS}(v|u; S)$
$W_E((i, j); S) = |W_{CS}(i|j; S) - W_{CS}(j|i; S)|$
}

In this case we may have more than two sub-trees and two sub-skeletons. We divide the sub-skeletons and the sub-trees into the same number of some groups of sub-skeletons and sub-trees which are balanced as much as possible with respect to $W_{CS}(v|u; S)$ and $W_{CS}(p|q; T)$, where u is the middle node

and v is a middle-connected node of the skeleton, and q is the middle node and p is a middle-connected node of the tree. For each group of sub-skeletons, and its related group of sub-trees, we call the mapping procedure recursively.

Case 3. In this case the tree has a middle node, but the skeleton does not have any middle node. The skeleton is divided into two sub-skeletons and the weights of the edges of the sub-skeletons are updated as in Case 1. The tree is divided into two or more sub-trees and the weights of the edges of the sub-trees are updated as in Case 2. We record in the mapping list that the middle node of the tree is mapped onto the middle point of the related middle edge of the skeleton. If there are more than two sub-trees, we divide the sub-trees into two groups of sub-trees which are balanced as much as possible with respect to $W_{CS}(v|u;T)$, where u is the middle node and v is a middle-connected node of the tree. The mapping procedure is called recursively for each group of sub-trees and the related sub-skeleton.

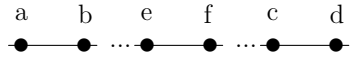

Fig. 2. Proof of lemma 1

Case 4. In this case the skeleton has a middle node, but the tree does not have any middle node. The tree is divided into two sub-trees and the weights of the edges of the sub-trees are updated as in Case 1. The skeleton is divided into two or more sub-skeletons and the weights of the edges of the sub-skeletons are updated as in Case 2. We substitute the middle edge of the tree with a path of length two whose extreme vertices are the endpoints of the middle edge and its internal vertex is a dummy vertex. We record in the mapping list that this newly added dummy vertex of the tree is mapped onto the middle node of the skeleton. If there are more than two sub-skeletons, we divide the sub-skeletons into two groups of sub-skeletons which are balanced as much as possible with respect to $W_{CS}(v|u;S)$, where u is the middle node and v is a middle-connected node of the skeleton. The mapping procedure is called recursively for each group of sub-skeletons and the related sub-tree.

Definition 6. For a tree T and its two nodes i and j, let $PathSet((i,j);T)$ be the set of all the nodes of the tree excluding i and j which lie on the path between nodes i and j. This definition is also applicable to the skeletons.

Lemma 1. If a skeleton S (tree T) has more than one middle edge, then these edges share an endpoint which is called the middle node.

Proof. We prove the lemma for skeleton S, a similar proof applies to tree T. Suppose edges (a,b) and (c,d) are two middle edges of skeleton S, as shown in Figure 2. If these two middle edges share an endpoint, the lemma is proved.

Otherwise, there is at least one edge (e, f) which lies on the path between nodes b and c (see Figure 2). Two cases are possible, case I in which $W_{CS}(e|f;S) \geq W_{CS}(f|e;S)$ and case II in which $W_{CS}(e|f;S) < W_{CS}(f|e;S)$. We prove the lemma for case I, the proof is similar for case II. From the definition we have

$$W_{CS}(c|d;S) \geq W_{IA}(c;S) + \sum_{i \epsilon PathSet((c,f);S)} W_{IA}(i;S) + W_{IA}(f;S) + W_{CS}(e|f;S)$$

$$\Longrightarrow W_{CS}(c|d;S) > W_{CS}(e|f;S) \tag{1}$$

$$W_{CS}(f|e;S) \geq W_{IA}(f;S) + \sum_{i \epsilon PathSet((f,c);S)} W_{IA}(i;S) + W_{IA}(c;S) + W_{CS}(d|c;S)$$

$$\Longrightarrow W_{CS}(d|c;S) < W_{CS}(f|e;S) \tag{2}$$

From relations (1) and (2) we have

$$W_{CS}(c|d;S) - W_{CS}(d|c;S) > W_{CS}(e|f;S) - W_{CS}(f|e;S) \geq 0$$

$$\Longrightarrow |W_{CS}(c|d;S) - W_{CS}(d|c;S)| > |W_{CS}(e|f;S) - W_{CS}(f|e;S)|$$

$$\Longrightarrow W_E((c,d);S) > W_E((e,f);S)$$

This contradicts the assumption that edge (c, d) is a middle edge. □

Step F. Removing possible crossings between the edges of the tree and the edges of the polygon

Before applying the SA method, all the nodes of the tree which are included in the mapping list are placed at the related points of the skeleton. Let *the closest located node* of a tree node be an already located node of the tree with the shortest graph-theoretic distance from the given tree node among all the previously located nodes of the tree. The remaining unlocated nodes of the tree are placed at the locations of their closest located nodes. After all the nodes of the tree are initially located, if the given polygon is non-convex there is the possibility of crossing between the edges of the tree and the border of the polygon. If this is the case, we remove these crossings by introducing some dummy nodes and bending the crossing edges of the tree. By applying rounding or truncation we obtain integer coordinates. The resulting configuration is the initial configuration of the SA method.

Step G. Drawing the tree using the SA method

As the final step of our algorithm, we use the SA method to draw the tree inside the given polygon. We try to keep the nodes of the tree as close as to their corresponding points of the skeleton by considering some new virtual fixed nodes and virtual edges. Our algorithm guides the SA method, so our drawing

results have fewer edge crossings and show more symmetry than the results of the algorithm introduced in [9], the SA algorithm.

4 DRAWING RESULTS

In this section, we compare the drawing results of our algorithm to those of the SA algorithm [9] by illustration of some examples. We also present some drawings of free trees by our algorithm on a 2D 480×640-grid which is bounded by some convex, rectilinear and concave polygons.

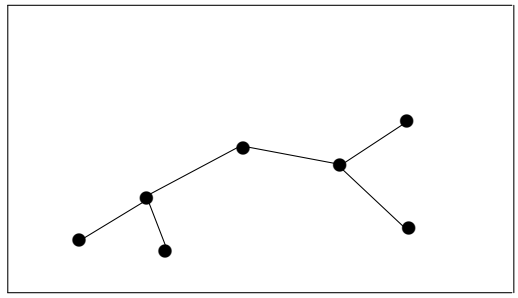

Fig. 3. Drawing of a 7-node complete binary tree by the SA algorithm

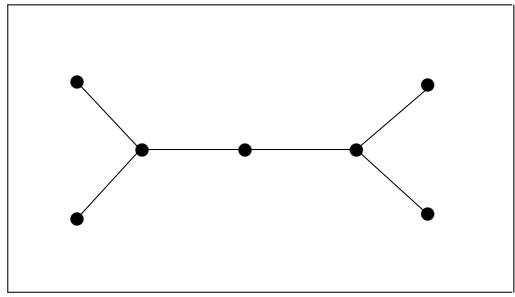

Fig. 4. Drawing of a 7-node complete binary tree by our algorithm

We use the same parameters and factors for the cost function of the SA method as is used by the SA algorithm. Figures 3 and 4 show the drawings of a complete binary tree that consists of seven nodes by the SA algorithm and by our algorithm, respectively. The size of the bounding rectangle is 100×200. As can be seen, our drawing seems nicer than the drawing of the SA algorithm. This is due to the symmetry of our drawing achieved using the geometrical properties of the bounding polygon by our algorithm.

The drawing of a complete binary tree with 63 vertices by the SA algorithm, inside a square, is shown in Figure 5. As can be seen, although the tree is planar its drawing by the SA algorithm is not planar. The drawing of this tree by our algorithm is shown in Figure 6. The size of the bounding square in these examples is 400×400. Our algorithm divides the given tree into some clusters of nodes by means of our mapping procedure, and distributes the nodes on different parts of the given polygon, and because of this, our algorithm usually has fewer edge crossings than the SA algorithm.

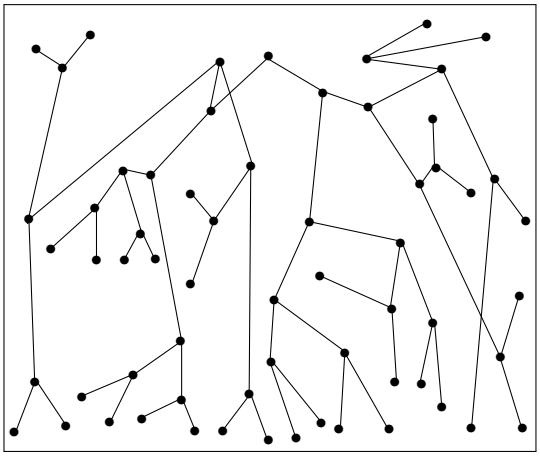

Fig. 5. Drawing of a 63-node complete binary tree by the SA algorithm inside a square

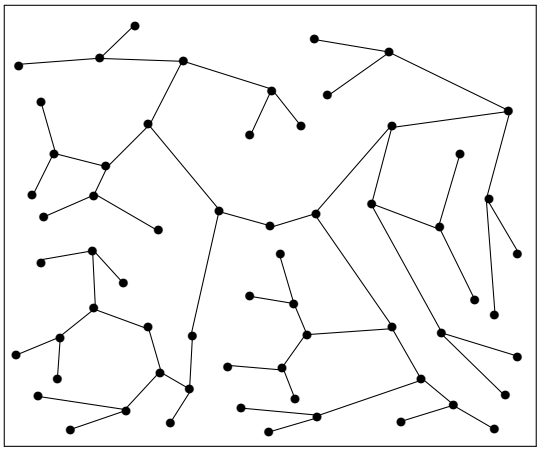

Fig. 6. Drawing of a 63-node complete binary tree by our algorithm inside a square

Figure 7 shows the drawing of a complete binary tree with 63 vertices by our algorithm inside a convex polygon. The size of the bounding square which includes

the given convex polygon is 400×400. Figures 8, 9 and 10 illustrate the drawings of a 31-node complete binary tree inside U-shaped, T-shaped, and S-shaped rectilinear polygons, respectively by our algorithm. The sizes of the bounding rectangles are 300×400, 200×300 and 300×300, respectively.

As the two final examples, the drawings of a complete binary tree with 31 vertices by our algorithm inside a W-shaped polygon and a NM-shaped polygon are shown in Figures 11 and 12, respectively. The sizes of the bounding rectangles are 200×400 and 200×600, respectively.

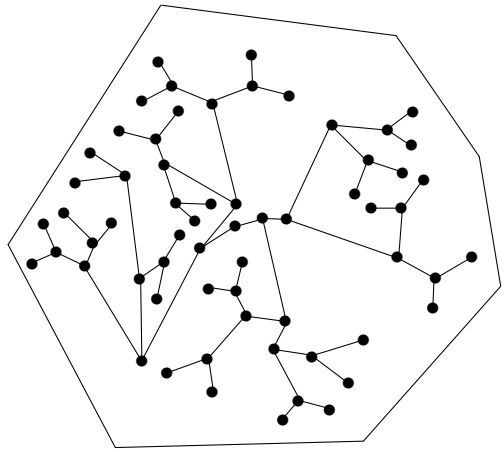

Fig. 7. Drawing of a 63-node complete binary tree by our algorithm inside a convex polygon

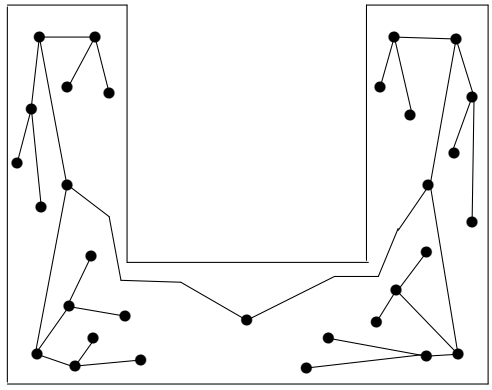

Fig. 8. Drawing of a 31-node complete binary tree by our algorithm inside a U-shaped rectilinear polygon

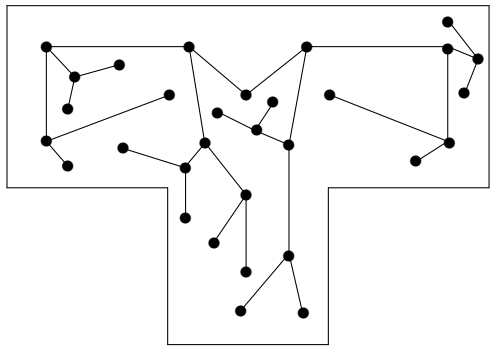

Fig. 9. Drawing of a 31-node complete binary tree by our algorithm inside a T-shaped rectilinear polygon

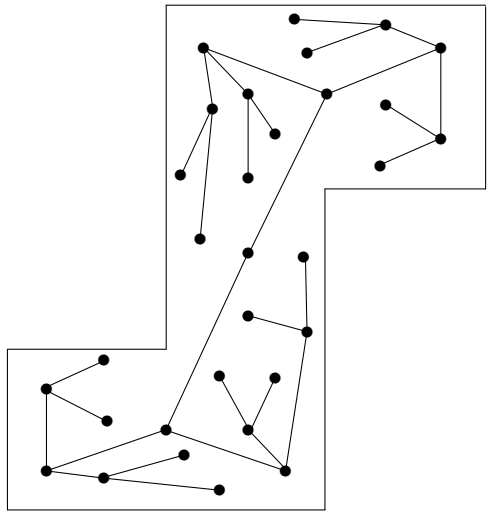

Fig. 10. Drawing of a 31-node complete binary tree by our algorithm inside a S-shaped rectilinear polygon

5 CONCLUSION

In this paper we introduced a new algorithm that employs the straight skeletons of polygons and the simulated annealing method to draw free trees inside general simple polygons. The drawing results show that our algorithm draws trees nicelier than the previous algorithms, with respect to the aesthetics that are mentioned at the introduction section, even for relatively large trees. This paper is the first attempt to develop algorithms which draw graphs on plane regions bounded by simple polygons.

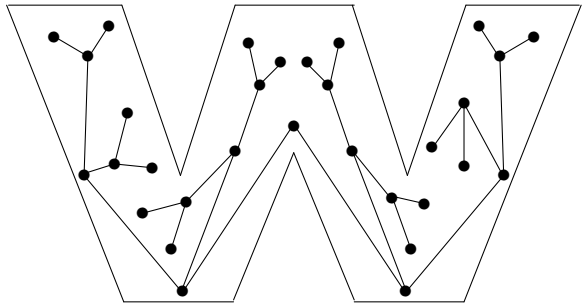

Fig. 11. Drawing of a 31-node complete binary tree by our algorithm inside a W-shaped polygon

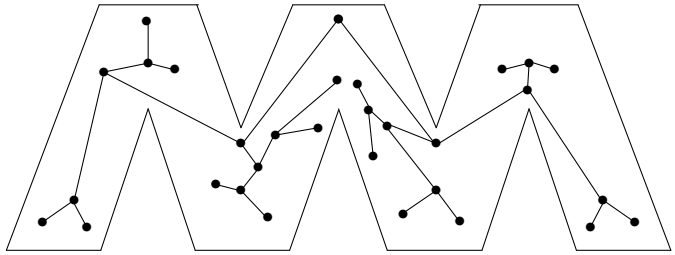

Fig. 12. Drawing of a 31-node complete binary tree by our algorithm inside a NM-shaped polygon

Acknowledgement

We would like to thank Oswin Aichholzer (`oaich@igi.tu-graz.ac.at`), Stepan Obdrzalek (`xobdrzal@fel.cvut.cz`) and Petr Felkel (`felkel@fel.cvut.cz`) for allowing us to use their straight skeleton implementations.

REFERENCES

[1] Aichholzer, O.—Aurenhammer, F.: Straight Skeletons for General Polygonal Figures in the Plane. In: Proeedings of 2nd Annual International Conference on Computing and Combinatorics, COCOON'96, Lecture Notes in Computer Science, Vol. 1090, 1996, pp. 117–126.

[2] Aichholzer, O.—Aurenhammer, F.—Alberts D.—Gartner, B.: Straight Skeletons of Simple Polygons. In: Proceedings of the 4th International Symposium of LIESMARS, LIESMARS'95, Wuhan/China, October 1995, pp. 114–124.

[3] Aichholzer, O.—Aurenhammer, F.—Alberts, D.—Gartner, B.: A Novel Type of Skeleton for Polygons. Journal of Universal Computer Science, Vol. 1, 1995, No. 12, pp. 752–761.

[4] Behrens, D.—Tolkiehn, R.—Barke, E.: Design Driven Partitioning. In: Proceedings of 2nd Asian and South Pacific Design Automation Conference, Asp-DAC'97, Chiba, Japan, January 1997.

[5] Bourke, P.: Calculating the area and centroid of a polygon. Available on: `http://www.swin.edu.au/astronomy/pbourke/geometry/polyarea/`, 1988.

[6] Brandenburg, F. J.: Graph Clustering I: Cycles of Cliques. In G. DiBattista (Ed): Graph Drawing, Proceedings of the Fifth International Symposium, GD'97, Rome, Italy, Lecture Notes in Computer Science, Vol. 1353, 1997, pp. 158–168.

[7] Chan, T. M.: A Near-Line Area Bound for Drawing Binary Trees. In: Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms, 1999, pp. 161–168.

[8] Chin, F.—Snoeyink, J.—Wang, C. A.: Finding the Medial Axis of a Simple Polygon in Linear Time. In: Proceedings of 6th Annual International Symposium on Algorithms and Computation, ISAAC'95, Lecture Notes in Computer Science, Vol. 1004, 1995, pp. 383–391.

[9] Davidson, R.—Harel, D.: Drawing Graphs Nicely Using Simulated Annealing. ACM Transactions on Graphics, Vol. 15, October 1996, No. 4, pp. 301–331.

[10] Eades, P.–Feng, Q. W.: Multilevel Visualization of Clustered Graphs. In: S. C. North (Ed): Graph Drawing, Proceedings of the 4th Symposium, GD'96, Lecture Notes in Computer Science, Vol. 1190, 1996, pp. 101–112.

[11] Eades, P.—Feng, Q. W.—Lin, X.: Straight-Line Drawing Algorithms for Hierarchical Graphs and Clustered Graphs. In: S. C. North (Ed): Graph Drawing, Proceedings of the 4$^{\text{th}}$ International Symposium, GD'96, Lecture Notes in Computer Science, Vol. 1190, 1996, pp. 113–128.

[12] Eades, P.—Feng, Q. W.—Nagamochi, H.: Drawing Clustered Graphs on an Orthogonal Grid. Journal of Graph Algorithms and Applications, Vol. 3, 1999, No. 4, pp. 3–29.

[13] Edachery, J.—Sen, A.—Brandenburg, F. J.: Graph Clustering Using Distance-K Cliques. In: Proceedings of the 7$^{\text{th}}$ International Symposium on Graph Drawing, GD'99, Lecture Notes in Computer Science, Vol. 1731, September 1999, pp. 98–106.

[14] Eppstein, D.—Erickson, J.: Raising Roofs, Crashing Cycles, and Playing Pool: Applications of a Data Structure for Finding Pairwise Interactions. In: Proceedings of the 14th ACM Symposium on Computational Geometry, SoCG'98, Minneapolis, 1998, pp. 58–67.

[15] Felkel, P.—Obdrzalek, S.: Straight Skeleton Implementation. In: Proceedings of Spring Conference on Computer Graphics, 1998, pp. 210–218.

[16] Purchase, H.: Which Aesthetic Has the Greatest Effect on Human Understanding? In G. DiBattista (Ed): Graph Drawing, Proceedings of the Fifth International Symposium, GD'97, Rome, Italy, Lecture Notes in Computer Science, Vol. 1353, 1997, pp. 248–261.

[17] Roxborough, T.—Sen, A.: Graph Clustering Using Multiway Ratio Cut. In G. DiBattista (Ed): Graph Drawing, Proceedings of the Fifth International Symposium, GD'97, Rome, Italy, Lecture Notes in Computer Science, Vol. 1353, 1997, pp. 291–296.

[18] Sablowski, R.—Frick, A.: Automatic Graph Clustering. In: S. C. North (Ed): Graph Drawing, Proceedings of the 4th International Symposium, GD'96, Lecture Notes in Computer Science, Vol. 1190, 1996, pp. pp. 396–400.

[19] Tan, X.—Tong, J.—Tan, P.—Park, N.—Lombardi, F.: An Efficient Multi-Way Algorithm for Balanced Partitioning of VLSI Circuits. In: Proceedings of IEEE International Conference on Computer Design, ICCD'97, 1997, pp. 608–613.

Alireza Bagheri received the B.S. and M.S. degrees in computer engineering from Sharif University of Technology (SUT) at Tehran. Currently he is a Ph.D. candidate in the computer engineering & IT department at Amirkabir University of Technology (AUT) at Tehran. His research interests include computational geometry, graph drawing and graph algorithms.

Mohammadreza Razzazi received the M.S. degree in computer scinece from Stanford University and the Ph.D. degree in computer science from the University of California at Santa Barbara. Currently he is an assistant professor in the computer engineering & IT department at Amirkabir University of Technology (AUT) at Tehran. His primary areas of research are computational geometry, robotics, and computer graphics.